Fondamenti di Informatica A
Laboratorio 10
Accesso al file system

Danilo Pianini
danilo.pianini@unibo.it
Mirko Viroli
mirko.viroli@unibo.it

Alma Mater Studiorum—Università di Bologna

22 maggio 2015

Operazioni preliminari

- Testo dell'esercitazione: `http://campus.unibo.it/190502/`
- **AVVISO:** per chi volesse sperimentare con i device file ed i comandi in stile Unix presentati a lezione, ha a disposizione una macchina virtuale Linux installata in `C:\VM\Courses\`
  1. Raggiungere la cartella `C:\VM\Courses\`
  2. Doppio click sul file `Courses.vbox` (icona blu)
  3. Si aprirà la finestra del gestore di macchine virtuali VirtualBox. Premere il tasto "Avvia"
  4. Attendere l'avvio della macchina
- La stessa macchina sarà utilizzata per il prossimo laboratorio

Linguaggio Assembly

Conoscere il linguaggio assembly non fa parte delle competenze che dovete acquisire per questo corso, ma è utile per capire come funziona la macchina sottostante (approfondirete in Calcolatori Elettronici!).

- Si consideri `sum.c`. Se ne comprenda il funzionamento, quindi si compili con:
  `gcc -S sum.c -o sum -Wall`
  Quali files sono stati prodotti?
- Si apra e si analizzi `sum.s`. Si cerchino delle corrispondenze fra il codice C e quello ASM:
  - Come sono organizzate le funzioni?
  - Come sono organizzati i loop?

Esempio di accesso al file system: `cat`

- Si osservi il contenuto di `cat_simple.c`. Cosa realizza?
- Una volta compreso il funzionamento, si compili e si esegua, ad esempio con
  `./cat_simple cat.c filter.c`
- Si osservi ora `cat.c`. In che modo differisce da `cat_simple.c`?
- Calcolare dinamicamente la dimensione del buffer da allocare ha vantaggi e svantaggi:
  - Evita di sprecare memoria
  - Consente di risolvere il problema di apertura di files più grandi del buffer predefinito senza la necessità fare multipli "giri" di lettura, risultando potenzialmente più veloce
  - Nel caso in cui un solo "giro" sia sufficiente a riempire il buffer statico, è potenzialmente più lento per via della `seek` / `rewind`
  - Rende il codice un po' più lungo

Esempio di accesso al file system: `cat`

- Si modifichi cat.c perché sia in grado di funzionare con più di un file in input, ossia:
  `./cat file1 file2 file3`
  restituisce in output
  `<contenuto di file1>`
  `<contenuto di file2>`
  `<contenuto di file3>`
- **SUGGERIMENTI:**
  1. Si crei una nuova funzione `void printFile(char *)` che, dato una stringa rappresentante il path di un file, lo stampa a schermo. Si noti che questa funzionalità è già offerta da `cat.c` all'interno del `main`, per cui sarà sufficiente spostare le linee di codice che vanno dall'apertura del file (linea 33) alla sua chiusura (linea 55) dentro la nuova funzione, avendo cura di cambiare il primo parametro della `fopen` opportunamente.
  2. Si testi la funzione chiamandola passando `argv[1]`: dovrebbe riprodurre il comportamento originale di `cat.c`.
  3. A questo punto, si aggiunga un ciclo `for` che scorra `argv` dal secondo elemento in poi e richiami al suo interno la funzione costruita sopra.
  4. Si elimini il messaggio di warning riguardante i parametri in eccesso.

Pipe

Pipe (simbolo |), come redirect (simboli <<, <, >>, >), è uno strumento che il terminale vi mette a disposizione per modificare da dove il vostro programma prende l'input e dove deve direzionare l'output.

- Pipe si utilizza per "concatenare" più programmi
- Lo standard output del programma alla sinistra della pipe viene messo in ingresso allo standard input del programma alla destra della pipe
- Qualunque cosa inviata allo standard output dal primo programma (ad esempio perché usa `printf`) sarà messa in ingresso allo standard input del secondo, che potrà accedervi, ad esempio, usando `gets`.

Filtro input-output, pipe, redirect

- Si osservi il contenuto di *filter.c*. Cosa realizza?
- Si compili e si testi il programma
- Si immagini che Twitter abbia messo a pagamento l'uso delle vocali: si modifichi `filter.c` affinché sostituisca a tutte le vocali della stringa in input la lettera Y, in modo da poter scrivere dei tweet il cui costo di pubblicazione sia zero.
- Si provi ad utilizzare il comando con la pipe:
  `echo look at the pipe! | filter`
  Come si comporta?
- Si provi ora ad utilizzare l'operatore di redirect per creare un nuovo file contenente il tweet:
  `echo look at the pipe! | filter > tweet`
  Cosa succede?

Salvataggio di dati

Si scriva un nuovo programma che:

- Legga due argomenti e li converta in numeri, interpretandoli come valore inizio e valore fine.
- Generi, ad intervalli di 0.01 a partire dal valore inizio e fino al valore fine, i valori della funzione coseno.
- Conservi questi valori all'interno di un apposito array
- Salvi tali valori in un file `data.log`. Il formato dei dati deve essere analogo a quello del file `data.log` fornito nei code examples.

SUGGERIMENTI — Si ricorda che:

- la funzione `double atof(const char*)` converte una stringa in double, si trova in `stdlib.h`
- la funzione `int fprintf(FILE *, const char *, ...)` in `stdio.h` consente di stampare su un descrittore di file del testo formattato (come `printf`, ma su qualunque file).
- esiste la funzione `double cos(double x)` di `math.h`.
- per compilare un sorgente che includa `math.h`, è necessario specificare la flag `-lm` a `gcc` (e.g. `gcc -lm -Wall programma.c -o programma`)
- Si osservi la slide 27 del pacchetto riguardante la gestione dei file!

Caricamento di dati

Si scriva un nuovo programma che:

- Legga un file `data.log` formattato come il file `data.log` allegato ai code examples.
- Inserisca i dati caricati all'interno di un apposito array (`double *`)
- Stampi a video il contenuto dell'array

SUGGERIMENTI — Si ricorda che:

- la funzione `int fscanf(FILE *, const char *, ...)` legge da uno stream qualsiasi una stringa formattata ed inserisce i risultati nelle aree di memoria puntate dagli argomenti aggiuntivi passati
- la funzione `void rewind(FILE *)` riporta il file descriptor passato all'inizio del file

Opzionali

Svolgete gli esercizi seguenti se avete completato tutta la precedente parte di esercitazione.

Modifica di cp: `mv`

- Si osservi il contenuto di `cp.c`. Cosa realizza?
- Sulla base di `cp.c` e `cat.c`, si realizzi un nuovo programma `mv.c` tale che:
  - Il programma invece di copiare il file lo sposti, ossia tale che la copia originale venga cancellata. Si utilizzi la funzione `int remove(const char *)` di `stdlib.h`, che dato un nome di file lo elimina. Si veda `http://www.cplusplus.com/reference/cstdio/remove/`
  - Il programma non utilizzi un array predimensionato come in cp.c, ma calcoli la dimensione del file (come in mv.c) e crei un array della dimensione appropriata. Si ricorda che in C per allocare una quantità di memoria nota solo a runtime è necessario l'uso della funzione `void* malloc (unsigned int size);`

replace

- Si osservi il contenuto del filtro `replace.c`. Cosa realizza?
- Sulla base di `replace.c`, si realizzi un nuovo programma `duplicate.c` tale che:
  - Il programma è un filtro che prende in ingresso un solo argomento, del quale considera il primo carattere
  - ogni volta che in ingresso il programma incontra il simbolo passato, deve duplicarlo.
  - **ESEMPIO:**
    `echo casa | ./duplicate a`
    deve avere come output:
    `caasaa`